

# **GUÍA METODOLÓGICA PARA LA REALIZACIÓN Y ENTREGA DE LOS LABORATORIOS DE ESTRUCTURAS DE DATOS Y ALGORITMOS**

*Texto Para Estudiantes*



# Tabla de Contenido

<b>Sección 0: Anotación del Docente</b>	<u>Pág. 3</u>
<b>Sección 1: Código de Ética</b>	<u>Pág. 5</u>
<b>Sección 2: Requisitos de Entrega</b>	<u>Pág. 7</u>
<b>Sección 3: Rúbricas de Evaluación</b>	<u>Pág. 9</u>
<b>Sección 4: Desarrollo de Procedimientos</b>	<u>Pág. 15</u>
<b>Sección 5: Guía GitHub</b>	<u>Pág. 39</u>

# **SECCIÓN 0: ANOTACIÓN DEL DOCENTE**

# Sección 0: Anotación del Docente

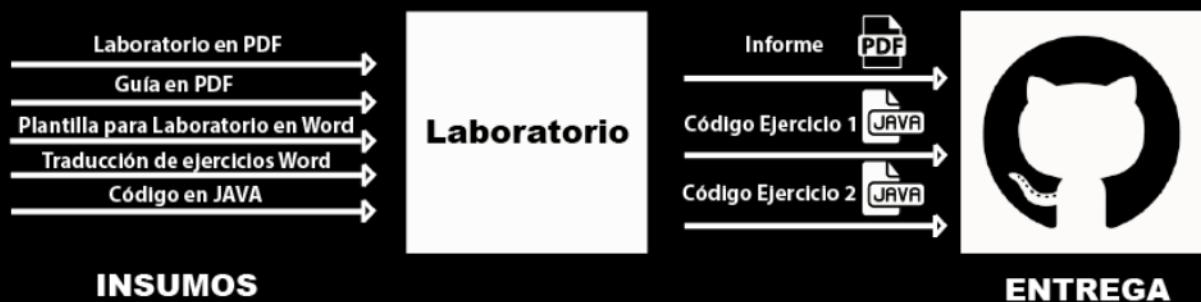
**Con este Texto – Guía se pretende facilitar la realización de la práctica de los laboratorios de la asignatura de Estructura de Datos y Algoritmos, y en general, ser útil en la comprensión de la metodología de trabajo para favorecer la obtención de los mejores resultados de los estudiantes**

## Intercambio de archivos

El **docente les entregará** a los estudiantes en GitHub, que contiene: Archivo PDF del taller, Archivo Word llamado "**plantilla-laboratorios.doc**", PDF de la lectura recomendada, Carpetas de código (Opcionalmente), Guía de laboratorios.

En GitHub, de manera opcional, también habrá traducciones al español de los ejercicios en línea.

Los alumnos, al realizar las actividades planteadas para cada laboratorio, **deberán entregar al docente** en GitHub, un Archivo PDF con el informe de laboratorio usando la plantilla y dos códigos en GitHub:



# **SECCIÓN 1: CÓDIGO DE ÉTICA**

# Sección 1: Código de Ética

## Código de Ética



Cada pareja de estudiantes **debe dar cuenta completa de todos y cada uno** de los ejercicios que entregan en el informe



Si utilizan código creado por ingenieros o estudiantes avanzados (amigos, familiares, etc.) u otras fuentes, -incluyendo las librerías de Java y el código que entregue el profesor-, según el tipo de citación correspondiente (código tomado de internet, código dentro de un código, código tomado del profesor).

Si el código que cita es el de un compañero del curso, deben referenciarlo, comparar la eficiencia en tiempo y memoria del código con respecto al suyo, y escribirlo en su informe

(Véase sección 4, numerales 4.15 y 4.16)



Si un estudiante no entiende el código que entregó, se manejará como fraude, de acuerdo con el Capítulo 1, Artículos 99 y 100 del Reglamento Estudiantil. Para leer más al respecto, visite: <http://bit.ly/2fm6kh2> o <http://bit.ly/2ge8vaV>

## **SECCIÓN 2: REQUISITOS DE ENTREGA**

# Sección 2: Requisitos de entrega

## Requisitos de Entrega

Lea atenta y completamente la presente Guía Metodológica. De su comprensión y el acatamiento de las instrucciones, dependerá el éxito de su calificación para los laboratorios que se realizarán en el transcurso del curso



**2.1** El laboratorio debe ser **desarrollado en parejas**



**2.2** El informe de laboratorio **debe ser entregado en .pdf**. No se recibirán informes en formato *.docx* o *.txt*

Para exportar a PDF utilizando Microsoft Word, haga clic en ‘Archivo’, ‘Guardar Como’, elija la ubicación de destino y el nombre del archivo. En “Tipo” escoja PDF y por último de clic en “Guardar”



**2.3** El **Código fuente** debe ser entregado en **GitHub**. No se reciben archivos en *.zip*

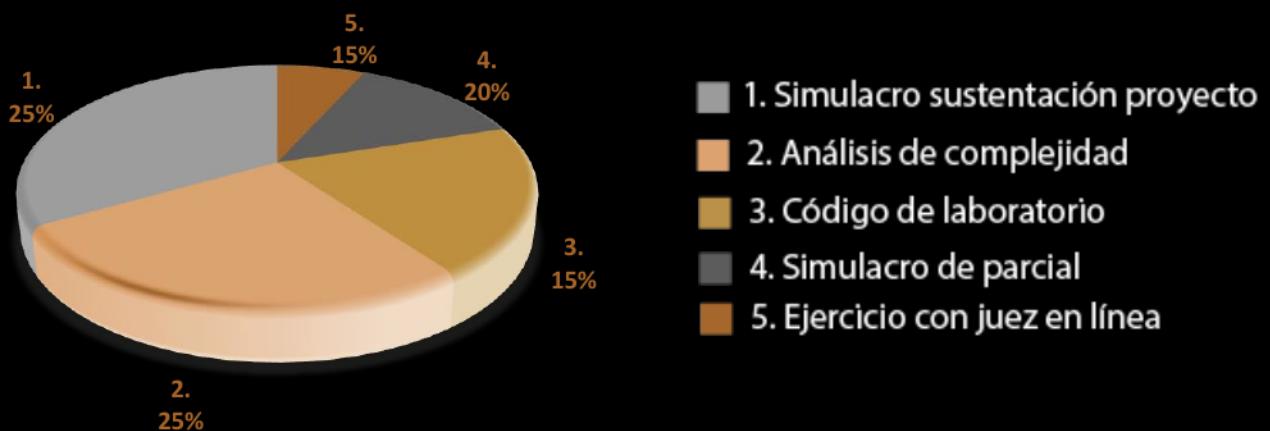


**2.4** El informe de laboratorio debe ser **entregado a través de GitHub**.

## **SECCIÓN 3: RÚBRICAS DE CALIFICACIÓN**

# Sección 3: Rúbricas de Evaluación

## Porcentajes y criterios de evaluación



### 3.1 Formatos de Entregas:

- Entrega del **informe de laboratorio** en formato **.pdf** y subirlo en GitHub
- Entrega del **código del Ejercicio en Línea** en “**.java**”, “**.cpp**” o “**.py**”

### 3.2 Para el Ejercicio en Línea (15%):

Entregar el **código fuente en .java, .cpp o .py** y **explicar brevemente en el informe PDF** dicho código, (entre 3 y 6 líneas de texto)

Aquí se evaluará lo siguiente:

- Constatación que el código **resuelva el problema sugerido**
- Implementación eficiente, es decir, que **cumpla con los** tiempos máximos sugeridos por el juez para el lenguaje de programación usado

# Sección 3: Rúbricas de Evaluación

## 3.3 Para el Código del Laboratorio (15%):

3.3.1 Entregar el **código de laboratorio en GitHub** y **explicar brevemente en el informe .pdf** dicho código y su funcionamiento (entre 3 y 6 líneas de texto)



Aquí se evaluará lo siguiente: **Correctitud del Código** y **Calidad del código**: identación, acoplamiento y cohesión



**NOTA:** Tanto para el **Ejercicio en Línea** como el **Código de Laboratorio**, si tienen **más del 40%** del código igual a otro grupo o hay **extracción de internet u otra fuente sin citación**, la **valoración será 0.0** en ambos casos. Es de carácter obligatorio hacer citación.

El **porcentaje de autenticidad** se verificará con la herramienta **Moss** de la **Universidad de Stanford**

## 3.4 Para el informe PDF (70%):

3.4.1 Dar las respuestas usando el archivo "**plantilla-laboratorios.doc**" que encuentra en el GitHub que el docente entrega. **No contiene aplicación de normas Icontec (25%)**



3.4.2 **Responder y sustentar** las preguntas que encuentra en cada taller de laboratorio. Use entre 3 y 6 líneas de texto para hacerlo, solo aquellas en que se le solicita explicar el **código del laboratorio**



Aquí se evaluará lo siguiente:

# Sección 3: Rúbricas de Evaluación

- Correctitud** de las **respuestas**
- Calidad** de la **argumentación**
- El **análisis de complejidad** y una **explicación de las variables** en las que quedó definida qué es “n”, “m”, etc... **(25%)**
- Simulacro Parcial** sobre **conocimientos teóricos (20%)**

**3.5 [Ejercicio Opcional] Lectura Recomendada (5%):** Cada laboratorio contiene una **lectura recomendada en formato .pdf**



Si deciden **hacer la lectura, pueden sumar puntos adicionales**, realizando las siguientes actividades:

- a) Resumen de lectura **(2.5%)**
- b) Realización de mapa conceptual **(2.5%)**



**NOTA:** Véase la **Sección 4**, numeral **4.19** de la presente guía, **un ejemplo concreto** para la realización de estas actividades

**3.6 [Ejercicio Opcional] Trabajo en Equipo y Progreso Gradual (5%):**

En este criterio se evalúa que hayan trabajado en equipo teniendo en cuenta que deberán:

- a) Entregar **copia** de todas las **actas de reunión**, con fecha, hora e integrantes que participaron
- b) El **reporte de git**, con los cambios en el código y quién hizo cada cambio, con fecha, hora e integrantes que participaron

# Sección 3: Rúbricas de Evaluación

- c) El reporte de cambios del informe de laboratorio que se genera Google docs o herramientas similares.



**NOTA 1:** Estas respuestas también van incluidas dentro del informe PDF



**NOTA 2:** Vea en la Sección 4.21 y 4.22 de la presente guía, **un ejemplo concreto** para la realización de estas actividades

**3.7 [Ejercicio Opcional]** Entregar el **código de laboratorio** con documentación en **Git** (5%)



**NOTA 1:** A Git no se le sube la documentación en HTML

**3.7.1** Para generar la **documentación en Git**, se recomiendan las siguientes herramientas:

Java	Javadoc	o Netbeans	o BlueJ
Python	Pydoc		
C++	Doxygen		

**3.7.2** Para generar la **documentación en JAVA**, se recomiendan las siguientes herramientas:

*Si usa Java, en la sección 3.1 se explica cómo escribir la documentación*

Tener en cuenta los siguientes **criterios de evaluación de la documentación Git** del código:

## Sección 3: Rúbricas de Evaluación

<b>Netbeans</b>	<b>Usar icono:</b> Martillo con escoba <b>No usar icono:</b> Flecha de Play	 
<b>BlueJ</b>	Control + J	

- Descripción de cada clase** con su propósito
- Descripción de cada método** incluyendo los parámetros que recibe y el valor que retorna
- Información general del código** en la clase principal, incluyendo autor y versión del código

**3.8 [Ejercicio Opcional]** Escribir el informe del laboratorio y el código en inglés (5%)

## **SECCIÓN 4: DESARROLLO DE PROCEDIMIENTOS**

# Sección 4: Desarrollo de Procedimientos

## 4.1 Cómo escribir la documentación Git de un código usando JavaDoc

Veamos en primer lugar **qué se debe incluir al documentar una clase y sus métodos:**

- a) Nombre de la clase, descripción general, número de versión, nombre de autores
- b) Documentación de cada constructor o método (especialmente los públicos), incluyendo: nombre del constructor o método, tipo de retorno, nombres y tipos de parámetros si los hay, descripción general, descripción de parámetros (si los hay), descripción del valor que devuelve



**NOTA:** Recuerde documentar con Doxygen si usan C++ o con Pydoc si usan Python

*Observe la tabla donde se muestran algunas de las palabras reservadas (tags):*

TAG	DESCRIPCIÓN	COMPRENDE
@author	Nombre del desarrollador.	Nombre autor o autores
@deprecated	Indica que el método o clase Descripción obsoleto (propio de versiones anteriores) y que no recomienda su uso.	
@param	Definición de un parámetro Nombre de parámetro un método, es requerido para descripción todos los parámetros método.	

# Sección 4: Desarrollo de Procedimientos

<b>@return</b>	Informa de lo que devuelve el método, no se aplica en constructores o métodos "void".	Descripción del valor retorno
	Referencia cruzada	
<b>@see</b>	Asocia con otro método o clase.	referencia (#método clase#método()); paquete.clase; paquete.clase#método()).
<b>@version</b>	Versión del método o clase.	Versión

Tomado de <http://bit.ly/2gcuoaH>

## 4.2 Cómo generar arreglos con valores aleatorios en Java

```
import java.util.Random;
public class test {
    public static void main(String[] args) {
        int size = 1000;
        int max = 5000;
        int[] array = new int[size];
        Random generator = new Random();
        for (int i = 0; i < size; i++)
            array[i] = generator.nextInt(max);
    }
}
```

# Sección 4: Desarrollo de Procedimientos

## 4.3 Cómo aumentar el tamaño del *heap* en Java

Si aparece **un error** “java.lang.OutOfMemoryError: Java heap space” error (64MB heap size), **a continuación las soluciones:**

- En **BlueJ**, no hemos podido solucionarlo, hay varios acercamientos en Internet, pero hasta ahora no funcionan.
- En **Netbeans**, mirar aquí cómo se hace: <http://bit.ly/2fYXHud>
- <http://javahowto.blogspot.com.co/2006/06/6-common-errors-in-setting-java-heap.html>

Lo que se muestra en el sitio web referenciado anteriormente, es **cómo aumentar el tamaño del *heap* con la directriz -Xmx**, la cual simplemente **dice cuál es el tamaño máximo** que puede tener el *heap*.

El ***heap* es donde se guardan los objetos al ser creados**, es decir, cuando ejecutan Objeto o = new Objeto(). Allí también se guardan los arreglos y otras variables. **La directriz -Xms es el tamaño inicial del *heap***. Asignándole 512MB (-Xms512m) no deberían tener ningún problema.



**NOTA:** Java limita el *heap* a 64MB y un arreglo de 100 millones ocupa 400 MB, por esto podría ser el error

## 4.4 Cómo aumentar el tamaño del *heap* y del *stack* en Java

Recuerden que **el *heap* es donde se guardan los objetos** y arreglos que uno crea con new. Si crean **arreglos muy grandes**, no duden en **incrementar el *heap*** así: **-Xmx512m**, por ejemplo

# Sección 4: Desarrollo de Procedimientos

Ahora, el **stack** es donde se guarda la **pila de la recursión**. Si usan recursiones que hagan **muchos millones de llamados**, no duden en **incrementar la pila** así: - **Xss512m**, por ejemplo



**NOTA:** SS = Stack Size

## 4.5 Cómo visualizar el montículo (*heap*) y el **stack**, y el consumo total de memoria de Java

Si tienen curiosidad o presentan **otros problemas**, les recomiendo esta **herramienta para visualizar gráficamente** lo que está pasando **con la máquina virtual de Java**: <http://visualvm.java.net/> . VisualVM **permite ver el montículo (*heap*)**, la **pila** y hace **una gráficas** a ver cómo se están llenando

Puede pasar que **se llene la RAM y el Sistema Operativo** pase parte de sus arreglos a Disco Duro, volviendo **lenta la ejecución**. Si es así, **revisar en el monitor de actividad** que ofrece su Sistema Operativo

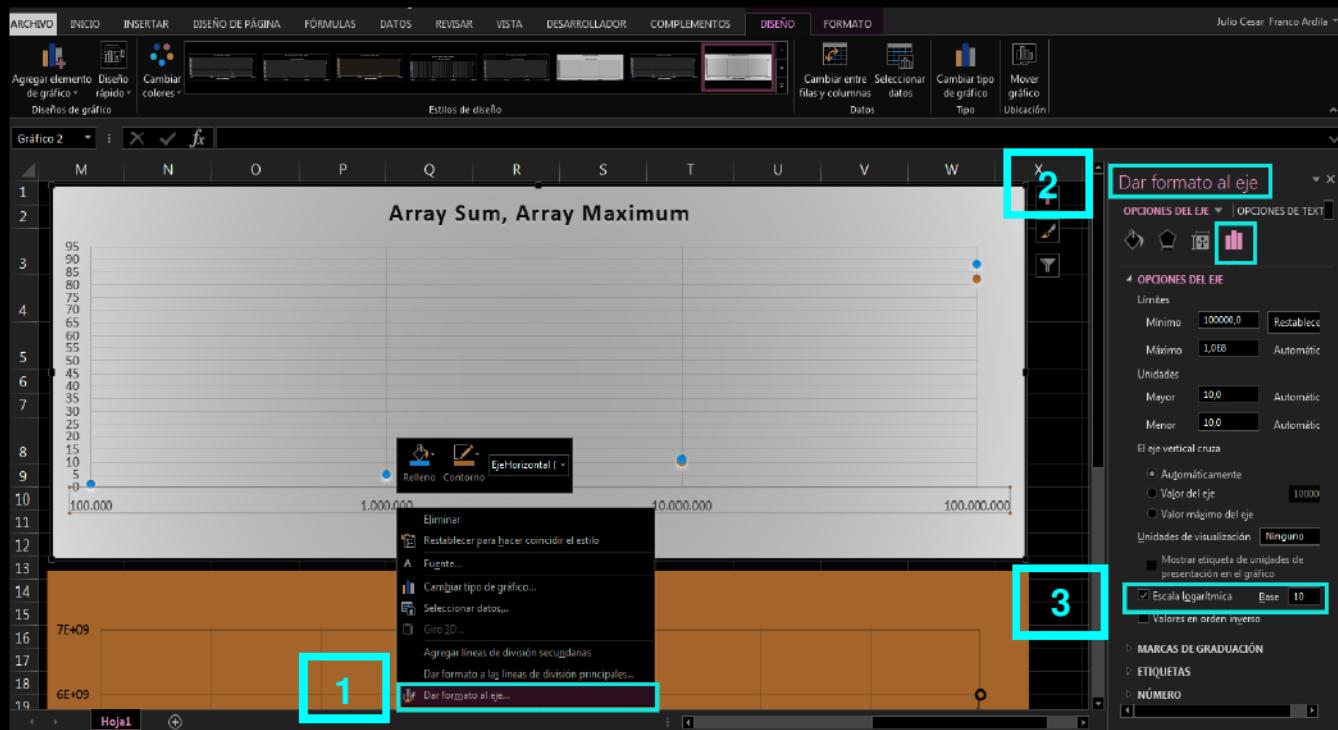
## 4.6 Cómo usar la escala logarítmica en Microsoft Excel 2013

Lo primero que hay que hacer es **dar clic derecho en el eje X de la gráfica** (por ejemplo, en algún número del eje x). Luego, **clic izquierdo en la opción "Dar formato al eje..."**.

En la versión de Excel 2013, inmediatamente en la parte derecha de la pantalla, **aparecerá un cuadro de opciones del formato del eje**, en el que se podrá **configurar el valor inicial y final** que tendrá el eje, entre otros. Antes de la sección "**Marcas de Graduación**", en negrilla aparece "**Escala Logarítmica**", **en base 10**. Sólo es necesario para el eje X debido a que los valores de éste eje son potencias de 10

# Sección 4: Desarrollo de Procedimientos

Observe gráficamente el procedimiento:



## 4.7 Cómo calcular el tiempo que toma un código en ejecutarse en Java

```
long startTime = System.currentTimeMillis();
// ... do something ...
long estimatedTime = System.currentTimeMillis() -  
startTime;
```

## 4.8 Cómo definir una clase Pareja en Java

En Java, **no es posible utilizar tuplas**, por ejemplo, (1,2) o (2,3), como sucede en lenguajes dinámicamente tipados como Python o Ruby. **Para poder guardar dos valores en un solo nodo de una lista o para poder retornar dos valores**, es necesario **crear un objeto de una clase** que llamaremos **Pareja**.

# Sección 4: Desarrollo de Procedimientos

Para poder **guardar el vértice de destino y el peso** en una lista de adyacencia, podemos **crear una lista de listas de parejas**:

```
LinkedList<LinkedList<Pareja>> listaDelistas = new  
LinkedList<LinkedList<Pareja>>();
```

**La clase pareja se define de la siguiente forma:**

```
class Pareja{  
    public int peso;  
    public int vertice;  
}
```

Otra alternativa es usar `javafx.util.Pair`

## 4.9 Cómo hacer una lista de Neveras

En Java, si se quiere **representar una nevera con su descripción y código**, una forma común es **hacer una clase Nevera** y luego **crear una lista de Neveras**.

**No es posible guardar dos objetos en el nodo de una lista**, así que uno puede hacer una **clase Nevera** (que contiene dos cosas) y luego hacer una **lista donde cada elemento es una Nevera**.

```
class Nevera  
{  
    public String descripcion;  
    public int codigo;  
    public Nevera(int c, String d) {descripcion = d; codigo = c;}  
  
}  
  
...  
  
LinkedList<Nevera> neveras = new LinkedList<Nevera>();
```

# Sección 4: Desarrollo de Procedimientos

## 4.10 Cómo usar clases abstractas en Java

Una **clase abstracta** es una clase que **funciona como una especificación para que otras clases la implementen**. Por ejemplo, en Java, la clase **AbstractList** sirve como especificación para diversos tipos de listas como son **ArrayList** y **LinkedList**.

La ventaja de la clase abstracta es que **el programador puede realizar un código genérico** que funcione para **ArrayList** y **LinkedList**. De igual manera, en este laboratorio, haremos lo mismo para dígrafos.

## 4.11 Cómo escribir la complejidad de un ejercicio en línea

### 1. Programa iterativo con una sola entrada, usualmente llamada “n”

#### a. Escribir el programa o algoritmo

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
        print(i+"*"+j+"="+i*j);
```

#### b. Etiquetar el programa:

```
for (int i = 1; i <= n; i++) // C1*n
    for (int j = 1; j <= n; j++) //C2*n2
        print(i+"*"+j+"="+i*j); //C3*n2
```

#### c. Calcular el T(n) sumando todas las anotaciones:

$$T(n) = (C2 + C3)n^2 + C1.n$$

#### d. Aplicar la notación O:

$$T(n) \text{ es } O((C2 + C3)n^2 + C1.n)$$

# Sección 4: Desarrollo de Procedimientos

e. Aplicar la regla de la suma:

$$T(n) \text{ es } O(C_2 + C_3)n^2$$

f. Aplicar la regla del producto:

$$T(n) \text{ es } O(n^2)$$

2. **Programa iterativo con varias entradas**, usualmente llamadas “n”, “m”... o “V”, “E”

a. Escribir el programa o algoritmo

```
for (int i = 1; i <= n; i++)
    for (int j = 1; j <= m; j++)
        print(i+"*"+j+"="+i*j);
```

b. Etiquetar el programa

```
for (int i = 1; i <= n; i++) // C1*n
    for (int j = 1; j <= m; j++) //C2*m.n
        print(i+"*"+j+"="+i*j); //C3*n.m
```

c. Calcular el  $T(n)$  sumando todas las anotaciones

$$T(n) = (C_2 + C_3)n.m + C_1.n$$

d. Aplicar la notación O

$$T(n) \text{ es } O(C_2 + C_3)n.m + C_1.n$$

e. Aplicar la regla de la suma

$$T(n) \text{ es } O(C_2 + C_3)n.m$$

# Sección 4: Desarrollo de Procedimientos

- f. Aplicar la regla del producto

$T(n)$  es  $O(n \cdot m)$

## 3. Programa recursivo

- a. Escribir el programa o algoritmo

```
SubProceso result <- Fibo( n )
    Definir result Como Entero;
    Si n <= 1 Entonces
        result <- n;
    Sino
        result<-Fibo (n-1)+Fibo (n-2);
```

- b. Etiquetar el programa

```
SubProceso result <- Fibo( n )
    Definir result Como Entero; // C1
    Si n <= 1 Entonces // C2
        result <- n; //C3
    Sino
        result<-Fibo (n-1) +Fibo (n-2); //C4+T(n-1)+T(n-2)
```

- c. Escribir el  $T(n)$  como una ecuación de recurrencia

$$T(n) = C_1 + C_2 + C_3, \text{ si } n \leq 1$$

$$T(n) = C_4 + T(n-1) + T(n-2), \text{ de lo contrario}$$

- d. Resolver la ecuación de recurrencia usando la teoría de polinomio característico, teorema maestro o <https://www.wolframalpha.com/>

$$T(n) = C \cdot 2^n + C'$$

# Sección 4: Desarrollo de Procedimientos

e. Aplicar la notación O

$$T(n) \text{ es } O(C \cdot 2^n + C')$$

f. Aplicar la regla de la suma

$$T(n) \text{ es } O(2^n)$$

g. Aplicar la regla del producto

$$T(n) \text{ es } O(2^n)$$

## 4.12 Cómo resolver ecuaciones de recurrencia lineales y no lineales

Hay 2 grandes tipos de ecuaciones de recurrencia.

**Las lineales tipo  $T(n) = T(n-1) + C$  y las no lineales tipo  $T(n) = T(2n/3) + n$ .**

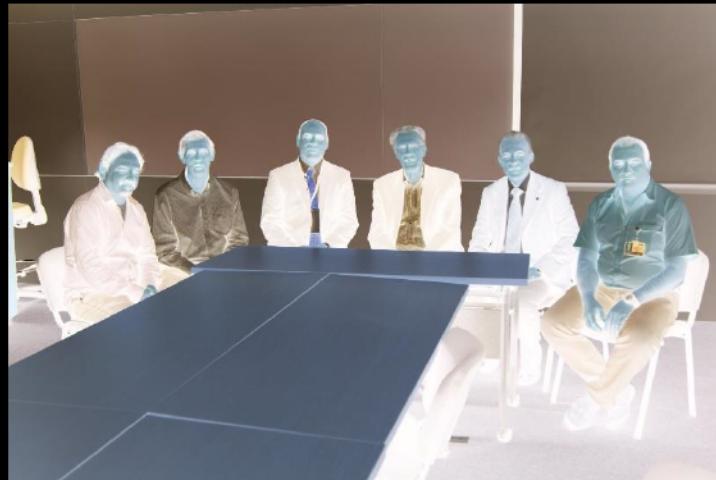
Las primeras se resuelven por expansión (también llamado, inducción) y las no lineales son más truculentas.

Dos libros recomendados, que pueden encontrarlos en Biblioteca, para estudiar este tema son:

- Thomas Cormen, Introduction to Algorithms (3th edition), 2009. Chapter 4
- John Hopcroft et al., Data Structures and Algorithms, 1983. Chapter 9

Sobre los autores, Cormen es un ilustrado de la complejidad en MIT y Hopcroft en Cornell. Hopcroft ha visitado Eafit y hay gran aprecio por él, pero el libro no está tan actualizado.

# Sección 4: Desarrollo de Procedimientos



**Pie de foto:** Hopcroft y su visita a EAFIT en el marco del Congreso Latinoamericano de Estudios en Informática (CLEI) del año 2012.

Finalmente, y no está de más, "los chinos" hacen muy buena explicación de complejidad, incluyendo las ecuaciones de recurrencia lineales, pero no entran en las no lineales en:

- R.C.T Lee et al., Introducción al análisis y diseño de algoritmos, 2005. Capítulo 2.
- Si quieren un resumen en Español del capítulo 4 de Cormen, léanlo en <http://bit.ly/2jWx4si>

## 4.13 Cómo usar Scanner o *BufferedReader*

Para la **lectura de datos por entrada estándar** en Java existen **dos formas** sencillas, una con la **clase Scanner** y otra con la **clase BufferedReader**. Para conocer más visite: <http://bit.ly/2ge1maJ>

## 4.14 Cómo hacer pruebas unitarias en BlueJ usando JUnit

# Sección 4: Desarrollo de Procedimientos

Las **pruebas unitarias sirven para probar el funcionamiento de un módulo de código**, por ejemplo, de un método de una clase.

Veamos, para una clase Sumador, que tiene un método que suma dos enteros, podemos diseñar pruebas, por ejemplo, que si se llama sumar (2,2) retorne 4, si se llama sumar (2,1) retorne 3, etc.

Para **hacer esto en BlueJ**, creamos una clase **SumadorTest** que sea de tipo **UnitTest**, representadas de color verde. Después, escogemos la opción “**create Test method**” en la clase **SumadorTest**.

Esto nos permite “**grabar**” una **secuencia de pasos**, obtener **un resultado** y decir **cuál debe ser el valor** de ese resultado. Como un **ejemplo**, podemos crear una instancia de la **clase Sumador**, luego **llamar el método suma** con parámetros (2,2) y el test debe **verificar** que **el valor de retorno** sea 4.

Así mismo podemos **hacer varios tests** para los otros métodos de la clase Sumador. Una vez estén hechos los tests, podemos **llamar a cada test independiente o la funcionalidad TestAll** que llama todos los tests.

**Las pruebas** son útiles para **verificar** que cuando uno **cambie el código** de su programa, **los métodos** sigan teniendo el **comportamiento esperado**.

Además, las pruebas **se deben diseñar incluso antes de escribir los métodos**, de la forma que se puedan **verificar** desde la **primera versión** que uno tenga de cada método de cada clase

# Sección 4: Desarrollo de Procedimientos

Finalmente, **una buena práctica es hacer tests para cada método por separado**, antes de hacer tests que involucren el llamado de varios métodos



## 4.15 Cómo compilar pruebas unitarias en Eclipse y en Netbeans

**Eclipse:** Clic derecho en el proyecto Eclipse y navegar de la siguiente forma: Properties, Java Build Path, Libraries, Add Library, Junit.

**Netbeans:** Click derecho en el proyecto del Laboratorio, en el panel *Projects*. Seleccionar el menú *Properties*. Seleccionar la categoría *Libraries* del panel *Categories*. Seleccionar la pestaña *Compile*. Hacer click en el botón *Add Library*. En el cuadro de diálogo *Add Library*, seleccionar la librería *JUnit 4*. Hacer click en el botón *Add Library*.

# Sección 4: Desarrollo de Procedimientos

## 4.16 Cómo reportar código tomado de Internet

### 1. Para **citar código** en un reporte, **se recomienda este formato:**

<author(s) names> (<date>) <title of program/source code> (<code version>) [<type>]. Web address or publisher.

**Por ejemplo,**

Smith, J (2011) GraphicsDrawer source code (Version 2.0) [Source code].  
<http://www.graphicsdrawer.com>

### 2. Para **citar código dentro de un código**, se recomienda este formato

/ Title: <title of program/source code>  
Author: <author(s) names>  
Date: <date>  
Code version: <code version>  
Availability: <where it's located>

/

**Por ejemplo,**

/  
Title: GraphicsDrawer source code  
Author: Smith, J  
Date: 2011  
Code version: 2.0  
Availability: <http://www.graphicsdrawer.com>

/

**Tomado de <http://bit.ly/2fm2zYI>**

# Sección 4: Desarrollo de Procedimientos

## 4.17 Cómo reportar código tomado de un profesor

Igual que para el **código tomado de internet**, pero colocando el **nombre del profesor** como **autor**, versión del código 1.0, y disponibilidad <http://interactiva.eafit.edu.co>

**Por ejemplo,**

Toro, M (2016) Graphics Drawer source code (Version 1.0) [Source code].  
<http://interactiva.eafit.edu.co>

## 4.18 Respuestas del Simulacro de Parcial

Simulacro de Parcial tipo Prueba Saber Pro con preguntas de selección múltiple con única opción de respuesta. Aquí deberá responder usando el número de pregunta, seguido de la letra que contiene la opción de respuesta que considera apropiada

**Por ejemplo,**

- 1) a
- 2) d
- 3) c

## 4.19 Ejemplos para calcular la complejidad de un ejercicio de CodingBat

### 4.19.1 MaxSpan

Consider the leftmost and righmost appearances of some value in an array. We'll say that the "span" is the number of elements between the two inclusive. A single value has a span of 1. Returns the largest span found in the given array. (Efficiency is not a priority.)

# Sección 4: Desarrollo de Procedimientos

```
public int maxSpan(int[] nums) {  
    int max=nums.length;  
    int cont1=0;  
    int cont2=0;  
    for(int i=0;i<max;i++) {  
        if (nums[i]==nums[max-1]) {  
            cont2=max-i;  
            break;  
        }  
    }  
    for(int i=max-1;i>=0;i--) {  
        if (nums[i]==nums[0]) {  
            cont1=i+1;  
            break;  
        }  
    }  
    if (cont1>cont2) return cont1;  
    return cont2;  
}
```

En este algoritmo se encuentran dos ciclos, pero estos no están anidados, sino que se encuentran por aparte, es decir que como los ciclos es lo que nos importan y ninguno contiene al otro entonces la complejidad de este algoritmo es lineal.

$$T(n) = C_1 + C_2n + C_3n$$

$$T(n) = C_1 + C_4n$$

Aplicando las reglas del producto y de la suma:

$$O(n) = C_4n$$

$$O(n) = n$$

# Sección 4: Desarrollo de Procedimientos

## 4.19.2 Fix34

Return an array that contains exactly the same numbers as the given array, but rearranged so that every 3 is immediately followed by a 4. Do not move the 3's, but every other number may move. The array contains the same number of 3's and 4's, every 3 has a number after it that is not a 3 or 4, and a 3 appears in the array before any 4.

```
public int[] fix34(int[] nums) {  
    int max=nums.length;  
    int arr1[]=new int[max];  
    int arr2[]=new int[max];  
    for(int i=0;i<max;i++) {  
        if(nums[i]==3) {  
            arr1[i+1]=nums[i+1];  
            nums[i+1]=4;  
        }  
    }  
    for(int i=0;i<max;i++) {  
        if(nums[i]==4 && arr1[i]==0) {  
            for(int j=0;j<max;j++) {  
                if(arr1[j]!=0&&arr1[j]!=4) {  
                    nums[i]=arr1[j];  
                }  
            }  
        }  
    }  
    return nums;  
}
```

En este caso el algoritmo consta de 3 ciclos, y se puede ver que un ciclo esta anidado con el otro, lo cual vuelve la complejidad cuadrática, mientras que el otro ciclo si está totalmente aparte de estos dos y no afecta en nada la complejidad que se encontró.

# Sección 4: Desarrollo de Procedimientos

$$T(n) = C_1 + C_2n + C_3n^2$$

Aplicando las reglas del producto y de la suma:

$$O(n) = C_2n + C_3n^2$$

$$O(n) = C_3n^2$$

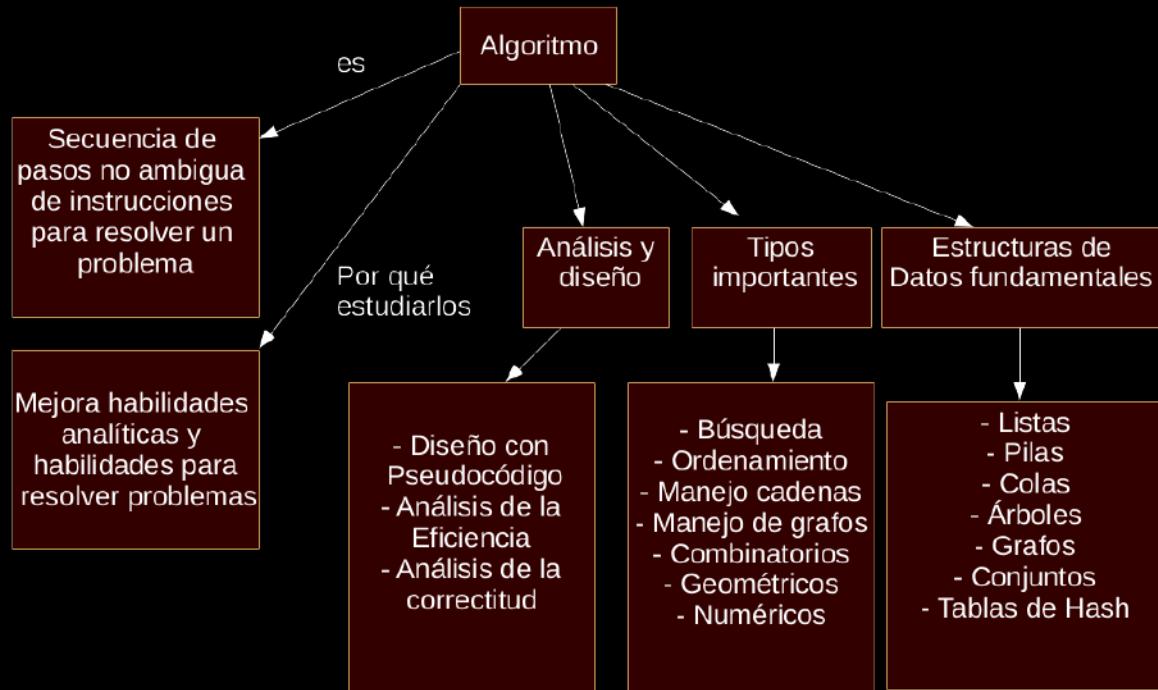
$$O(n) = n^2$$

## 4.20 Ejemplos para realización de actividades de las Lecturas Recomendadas

**4.20.1 Ejemplo de resumen:** Un algoritmo es una secuencia no ambigua de instrucciones para resolver un problema. Es importante estudiarlos porque mejora las habilidades analíticas para resolver problemas. Los tipos importantes son: de búsqueda, ordenamiento, manejo de cadenas, manejo de grafos, combinatorios, geométricos y numéricos. Las estructuras de datos fundamentales son: listas, pilas, colas, árboles, grafos, conjuntos y tablas de hash. Para diseñar un algoritmo se utiliza una notación llamada pseudocódigo. Para analizar los algoritmos existen dos tipos de análisis: de eficiencia y de correctitud.

## 4.20.2 Representación gráfica de los conceptos más importantes del artículo \*

# Sección 4: Desarrollo de Procedimientos



**NOTA:** La gráfica de conceptos o mapa conceptual, **no será un “copy paste”** que hagan desde la web. Deben **construir su propio mapa de conceptos** a partir de herramientas como las que encuentra en <https://cacoo.com/>

\*Ejemplo a partir de la lectura '*Introduction to the design and analysis of algorithms. Chapter 1*'

## Sección 4: Desarrollo de Procedimientos

### 4.21 Ejemplo de cómo hacer actas de trabajo en equipo usando Tablero Kanban

Integrante	Fecha	Hecho	Haciendo	Por Hacer
Mateo	01/08/2016	Implementé matrices de adyacencia		Probar Dijkstra sobre mi implementación
Catalina	01/08/2016	Implementó listas de adyacencia		Probar Dijkstra sobre mi implementación
Mateo	03/08/2016	Dijkstra funcionó	Revisar el trabajo de Catalina	Resolver ejercicios en línea
Catalina	03/08/2016	Dijkstra función	Revisar el trabajo de Mateo	Resolver ejercicios en línea
Mateo	06/08/2016	Resolver ejercicios en línea	Resolviendo las preguntas entre los dos	Hacer la lectura cada uno
Catalina	06/08/2016	Resolver ejercicios en línea	Resolviendo las preguntas entre los dos	Hacer la lectura cada uno

### 4.22 Cómo ver el historial de revisión de un archivo en Google Docs

1. Abre un documento, una hoja de cálculo, una presentación o un dibujo
2. Haz clic en **Archivo > Ver historial de revisión**
3. Haz **clic en una marca de tiempo** en el panel de la derecha para ver una versión anterior del archivo.

# Sección 4: Desarrollo de Procedimientos

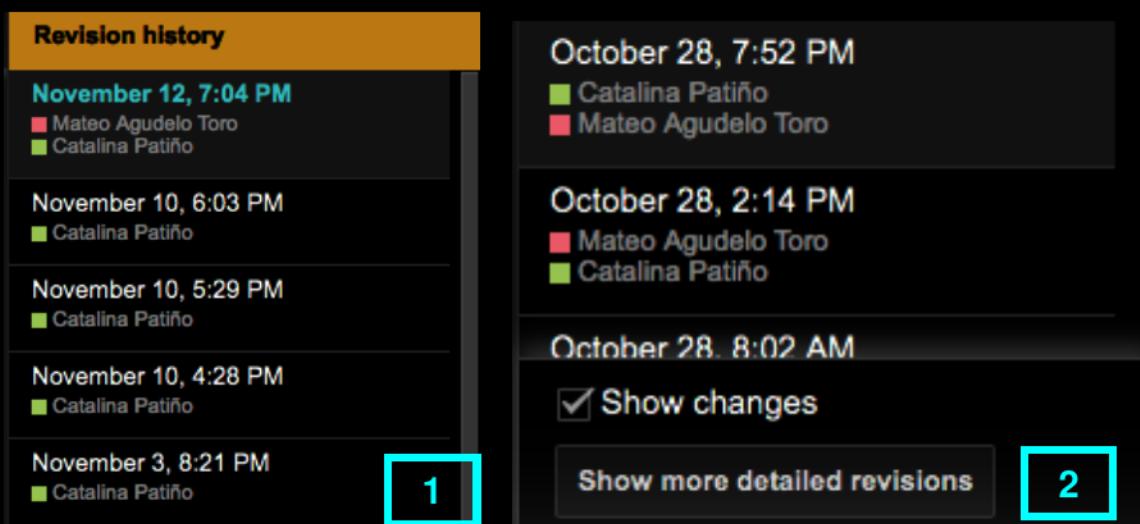
También verás quién ha editado el archivo **debajo de la marca de tiempo**. Las modificaciones que cada persona ha hecho **se muestran en el color** que aparece junto a su nombre

4. En la parte superior derecha, **usa las flechas para desplazarte** por la lista de cambios

**Nota:** El texto añadido se resalta en otro color y el eliminado aparece tachado

5. Para volver a la versión que estás viendo, haz clic en **Restaurar esta revisión**
6. Para volver a la versión actual del documento, **haz clic en la flecha hacia atrás** situada en la parte superior izquierda

*Observe gráficamente la Revisión del Historial:*



\*Tomado de <http://bit.ly/20vEuRa>

# Sección 4: Desarrollo de Procedimientos

## 4.23 Cómo generar el historial de cambios en el código de un repositorio que está en svn

Se hace ejecutando el siguiente comando:

```
svn log -v
```

Y Se obtiene algo así:

```
-----
-----
r33 | estudiante1 | 2012-02-28 16:10:41 +0600 (Вт, 28 фев 2012)
| 1 line
```

One more change

```
-----
-----
r32 | estudiante2 | 2011-12-27 17:37:31 +0600 (Вт, 27 дек 2011)
| 1 line
```

Cleanups

```
-----
-----
r31 | estudiante1 | 2011-12-27 17:29:00 +0600 (Вт, 27 дек 2011)
| 1 line
```

Purification

```
-----
-----
r30 | estudiante2 | 2011-10-19 16:23:52 +0600 (Ср, 19 окт 2011)
| 1 line
```

# Sección 4: Desarrollo de Procedimientos

Try fix FS #2

-----  
-----

r29 | estudiante1 | 2011-10-19 16:18:43 +0600 (Cp, 19 окт 2011)  
| 1 line

Si utilizan *svn tortoise*, se verá así

The screenshot shows the 'Log Messages' window of TortoiseSVN. The title bar reads 'Log Messages - C:\Projects\TortoiseSVN'. The window has a search bar at the top right labeled 'Messages, authors and paths'. Below it is a table with columns: Revision, Actions, Author, Date, Bug-ID, and Message. The table contains the following data:

Revision	Actions	Author	Date	Bug-ID	Message
7771	U	steveking	9:16:57 AM, Saturday, October 14, 2006		Remove obs
7769	U	steveking	8:49:05 AM, Saturday, October 14, 2006	314	Fix UI proble
7768	U	schaefer	8:02:57 AM, Saturday, October 14, 2006	306	Menu items v
7767	U	steveking	5:57:27 AM, Saturday, October 14, 2006	314	Fix problems
7766	U	steveking	1:39:45 PM, Friday, October 13, 2006		fix link to the
7764	U	simonlarge	1:28:10 PM, Friday, October 13, 2006		Mention the .
7763	U	luebbe	12:58:08 AM, Friday, October 13, 2006		Fix "Invalid C
7762	U	luebbe	12:55:11 AM Friday October 13 2006		Guি translation

# **SECCIÓN 5: GUÍA GITHUB**

# Sección 5: Guía GitHub

*Durante el curso de Estructuras de Datos y Algoritmos, se realizarán distintas actividades (Laboratorios, Talleres en clase y Proyectos) las cuales contarán en su mayoría con una gran cantidad de ficheros (entre código y documentos), los cuales deberán ser entregados por medio del repositorio de control de versiones GitHub.*

*Esta guía tiene como propósito enseñarle los conceptos básicos de este control de versiones y poder administrar sus entregas con éxito.*

*Lo invitamos a que explore en las funcionalidades del repositorio de control de versiones más usado del mundo.*



**NOTA:** Aquí le enseñaremos **cómo administrar GitHub** (y otros repositorios basados en *git*) por medio del uso del *bash*.

Para los **estudiantes que se reúsen al uso de este mecanismo**, también se darán algunos tips para el uso de GitHub por medio del *Browser*, en ese caso siga la **Sección GitHub Con Browser** de cada uno de los pasos de la guía.

# Sección 5: Guía GitHub

## Pre requisitos de la Guía

**Tengan en cuenta las siguientes consideraciones iniciales:**

- Tener una consola *bash*: Para las personas que no cuenten con distribuciones *Linux* (*Ubuntu*, *Debian*, *Fedora*, *RedHat*, etc.):
- Si usan arquitecturas basadas en *MacOS* la consola integrada cuenta usualmente con *git*.
- Si usa *Windows 10* puede acceder a una consola integrada basada en *Ubuntu* y usarla. En caso de que cuente con otra versión de *Windows*, también puede usar programas que emulen el ambiente *Linux* como *Cygwin*, entre otros, aunque no es recomendado.
- Conocer nociones mínimas del uso de ambientes *Linux*, *bash*: Usualmente son enseñadas en las primeras clases del curso de lenguajes.
- Git* es el protocolo y GitHub es un sitio web que usa el protocolo *Git*.
- Conocer los conceptos clave para el uso de repositorios basados en *Git*: puestos en esta misma guía

## Repositorios a usar

**Repositorio Plantilla:** Su uso será para permitir que el estudiante obtenga la estructura de directorios indicada para poder ingresar todas sus entregas de forma ordenada. La idea es importarlo a su repositorio personal.

- Si su materia es Estructura de datos y algoritmo 1 este es el link a ingresar:  
<https://github.com/mauriciotoro/ST0245-Plantilla>
- Si su materia es Estructura de datos y algoritmo 2 este es el link a ingresar:  
<https://github.com/mauriciotoro/ST0247-Plantilla>

# Sección 5: Guía GitHub

**Repositorio de Enunciados:** Es el repositorio donde se encuentran todos los enunciados, códigos y problemas que son parte de las actividades de la materia.

La idea es que sean visualizados y descargados desde el navegador web<sup>1</sup>, véase como una alternativa más elegante que descargar los enunciados de Eafit interactiva.

No está hecho para que sea clonado localmente en su máquina ya que continuamente serán actualizados.

- Si su materia es Estructura de datos y Algoritmos 1:  
<https://github.com/mauriciotoro/ST0245-Eafit>
- Si su materia es Estructura de datos y Algoritmos 2:  
<https://github.com/mauriciotoro/ST0247-Eafit>

## Conceptos Clave

**Control de Versiones<sup>2</sup>:** Se define como la gestión de todas las configuraciones de un producto [en nuestro caso, software]. Da la posibilidad de restaurar a versiones de archivos específicos o todo el repositorio.

**Commit:** Acción de ingresar todos los cambios existentes en el *Work Tree* al *HEAD*, usualmente se hacen junto un comentario que describa los cambios adicionados.

**Head:** *Branch*<sup>3</sup> que actualmente está siendo usada.

**Repositorio<sup>4</sup>:** Estructura de datos Jerárquica que almacena los ficheros que hacen parte de un proyecto.

**Push:** Ingresá todos los *Commits* que existen en el head al servidor de GitHub. En otras palabras, sube los cambios.

---

<sup>1</sup>Es por eso que los links presentados son para visualizar el repositorio via browser.

<sup>2</sup>Tomado de: [https://es.wikipedia.org/wiki/Control\\_de\\_versiones](https://es.wikipedia.org/wiki/Control_de_versiones)

<sup>3</sup>Rama. Concepto ampliamente tratado en GitHub y de mucha utilidad.

<sup>4</sup>Tomado de: [https://tortoisessvn.net/docs/nightly/TortoiseSVN\\_es/tsvn-basics.html](https://tortoisessvn.net/docs/nightly/TortoiseSVN_es/tsvn-basics.html)

# Sección 5: Guía GitHub

**Pull:** Actualiza la copia local del repositorio con los cambios que existan en el servidor de GitHub. En caso de que un mismo fichero haya sido modificado de manera local y además tenga cambios por actualizar, se generará una colisión<sup>5</sup>.

**Reset:** Quita todas las actualizaciones agregadas en el *WorkTree*.

**Working Tree:** Estructura donde se puede ver el estado<sup>6</sup> de los ficheros y/o directorios que están dentro de la carpeta del repositorio (la cual está bajo el control de versiones).

**Clonar (Clone):** Clonar un repositorio permite tener una copia local del mismo en su máquina. Si el usuario tiene permisos de edición, puede agregar, modificar y borrar los directorios y ficheros para luego actualizar esos cambios en el repositorio. Un repositorio puede ser clonado a partir de un link con protocolo HTTPS o por medio de una conexión SSH.

## Paso a paso en GitHub

### 0. Creando una cuenta en GitHub

0. En caso de tener una cuenta previamente, por favor vaya al Paso 7.
1. Ingrese a <https://github.com/>.
2. En el campo *username* recomendamos usar su nombre de usuario de Eafit, aunque es libre.
3. En email, escriba su correo de la Universidad

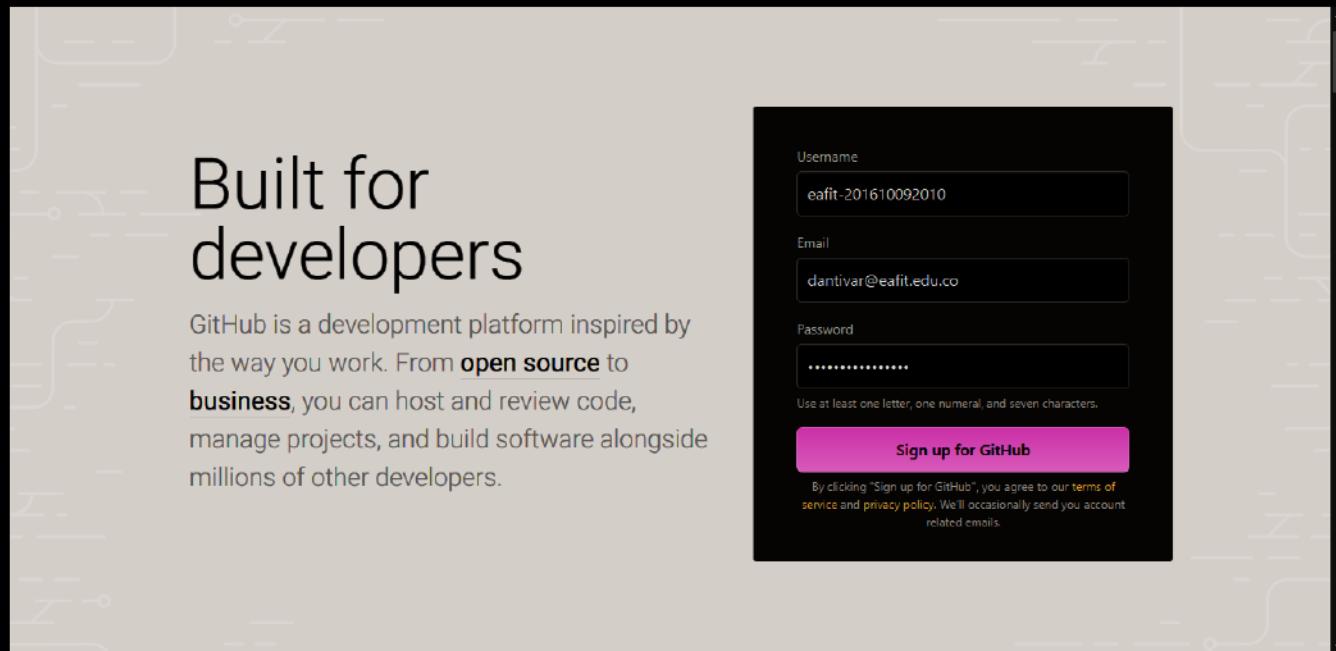
---

<sup>5</sup>Tema no tratado en esta guía

<sup>6</sup>Algunos ejemplos de estados son: new file, modified y renamed.

# Sección 5: Guía GitHub

4. En *password*, escriba la contraseña de su preferencia. **Importante:** Su contraseña debe tener al menos una letra, un número y 7 caracteres.
5. Haga click en *Sign up for Github*.



6. Complete la pequeña encuesta. (En la segunda pregunta, seleccionar *School projects*).

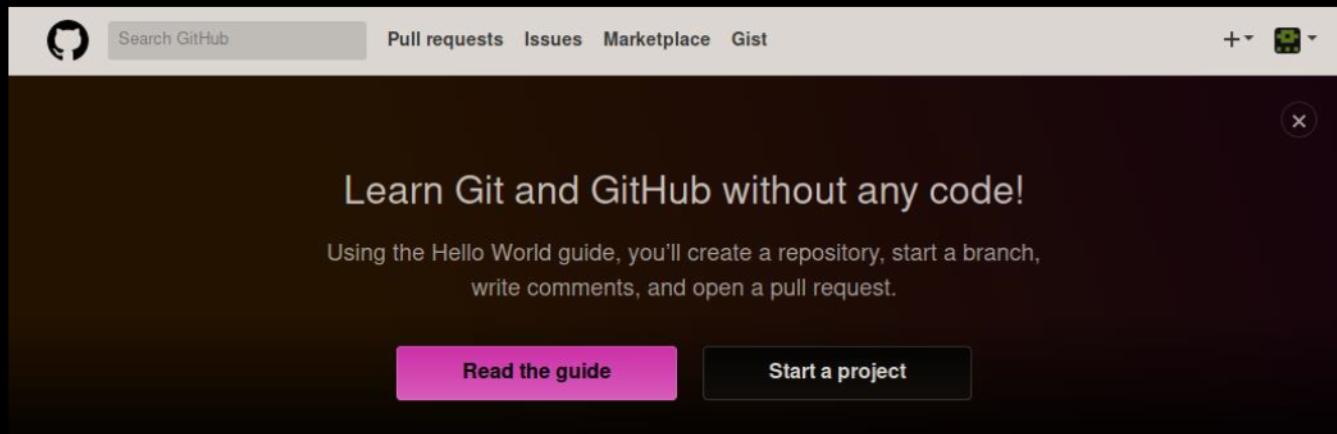
**Importante:** Recuerde verificar su correo electrónico ingresando a su cuenta de correo y siguiendo las instrucciones enviadas por *GitHub*

7. **Muy importante:** Por favor diligencie el siguiente formulario <http://bit.ly/2fxqsCF>

# Sección 5: Guía GitHub

## 1. Creando una cuenta en GitHub

1. Ingrese a *Github* con su cuenta personal. Vaya a la página principal y de clic en el botón “*Start a project*”



2. Escriba el nombre del repositorio, así:

- Curso de Estructuras de Datos y Algoritmos 1:  
ST0245-<numero de su grupo>
- Curso de Estructuras de Datos y Algoritmos 2:  
ST0247-<numero de su grupo>

# Sección 5: Guía GitHub

## Create a new repository

A repository contains all the files for your project, including the revision history.

Owner      Repository name  
 jarcil13 /

Great repository names are short and memorable. Need inspiration? How about [expert-octo-winner](#).

Description (optional)

 Public  
Anyone can see this repository. You choose who can commit.

 Private  
You choose who can see and commit to this repository.

Initialize this repository with a README  
This will let you immediately clone the repository to your computer. Skip this step if you're importing an existing repository.

Add .gitignore:  Add a license:  ⓘ

**Por ejemplo:** ST0245-032

Tenga en cuenta que el repositorio deberá ser **público**<sup>7</sup> y no deberá ser inicializado automáticamente por *GitHub*<sup>8</sup>

<sup>7</sup>De no ser público no será posible calificar sus trabajos

<sup>8</sup>Es decir, no seleccione la opción “Initialize this repository with a README”

# Sección 5: Guía GitHub

## 2. Inicializando su repositorio

Con el fin de poder organizar adecuadamente todas las entregas, hemos creado una estructura de directorios que usted deberá tener en su repositorio.

Si hizo correctamente el Paso 1, cuando ingrese a su repositorio deberá ver una pantalla parecida a esta:

The screenshot shows a GitHub repository page for 'jarcil13 / ST0245'. At the top, there are buttons for Watch (0), Star (0), Fork (0), and Insights. Below the header, there are tabs for Code, Issues (0), Pull requests (0), Projects (0), Wiki, Settings, and Insights. A large central box contains the following sections:

- Quick setup — if you've done this kind of thing before**  
or **HTTPS** **SSH** <https://github.com/jarcil13/ST0245.git>  
We recommend every repository include a [README](#), [LICENSE](#), and [.gitignore](#).
- ...or create a new repository on the command line**  

```
echo "# ST0245" >> README.md
git init
git add README.md
git commit -m "first commit"
git remote add origin https://github.com/jarcil13/ST0245.git
git push -u origin master
```
- ...or push an existing repository from the command line**  

```
git remote add origin https://github.com/jarcil13/ST0245.git
git push -u origin master
```
- ...or import code from another repository**  
You can initialize this repository with code from a Subversion, Mercurial, or TFS project.  
[Import code](#)

De clic en la opción “*Import Code from another repository*”. En la siguiente página ingrese el link del Repositorio *Plantilla*, asegúrese que en la opción de abajo esté el nombre del repositorio de esta materia.

# Sección 5: Guía GitHub

El link del repositorio a usar lo encontrará en la sección:

...[Repositorios a usar/repositorio plantilla] que se encuentra en esta guía.

Una vez termine el proceso ya tendrá su repositorio inicializado.

## Git con navegador web

Para agregar un colaborador (Usuario con permisos de *read/write* en su repositorio):

En la pantalla de inicio de su repositorio, haga clic en *Settings*. En parte izquierda encontrará un pequeño menú, haga clic en *colaborators*, agregue un colaborador por su nombre de usuario y haciendo clic en *Add collaborator*.

Recuerde agregar como colaborador a su pareja.



**NOTA:** Si trabajan en grupo, subir solo un trabajo al repositorio

## 3. Clonando su repositorio



**NOTA:** Ahora haremos una serie de modificaciones en el repositorio. De este paso en adelante, verá las indicaciones del *bash* y del *Browser*

# Sección 5: Guía GitHub

En su repositorio deberá verse similar a esta foto, si no es así, vuelva a los Pasos 1 o 2.

📁 laboratorios	Agregando estructura
📁 proyecto	Agregando estructura
📁 talleres	Agregando estructura

## Git con *Bash*

1. De click en la opción *Clone or Download*. Copie el link que se desplegará, asegúrese que use protocolo HTTPS.
2. Abra su consola y diríjase a la carpeta donde desee almacenar los datos de la materia.
3. Ingrese el comando:

```
$ git clone <link HTTPS del repositorio>
$ cd <nombre de su repositorio>
```

**PISTA:** Cuando no conozca la utilidad, sintaxis, opciones y demás información de algun comando en *git* siempre es útil consultar con `$ git <comando> --help`

Ahora veamos el estado de su repositorio (este comando será usado muy habitualmente durante toda la guía)

```
$ git status
```

Si observa una salida similar a esta, significa que tiene su repositorio ha sido clonado correctamente, podemos continuar.

# Sección 5: Guía GitHub

```
[juand@juanpc structureExample]$ git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
nothing to commit, working tree clean
```

4. Ahora inicialicemos las credenciales y realice los dos siguientes comandos

```
$ git config --global user.name "<Su Nombre>"  
$ git config --global user.email "<Su correo asociado a la cuenta GitHub>"
```

**Ejemplo:**

```
$ git config --global user.name "Pepito Perez Doe"  
$ git config --global user.email "pepiPD@eafit.edu.co"
```

5. Luego verifiquemos las credenciales:

```
$ git config --list
```

Deberá ver una salida parecida a esta, asegúrese que tenga los parámetros *name* y *email*:

```
juan@juan-VirtualBox:~$ git config --list  
user.name=Pepito Perez Doe  
user.email=pepiPD@eafit.edu.co
```

## 4. Agregando ficheros

### Git con *Bash*

Una vez estemos en el repositorio vamos a agregar un archivo llamado *TestFile.txt*

1. Primero creamos el archivo

```
$ touch TestFile.txt
```

# Sección 5: Guía GitHub

Observe como es indexado (en rojo) en el Work Tree

```
$ git status
```

**IMPORTANTE:** Ahora vamos a hacer el procedimiento para subirlo a *GitHub*. Este procedimiento es siempre igual para cualquier otro fichero o directorio.

2. Agregamos el fichero al HEAD

```
$ git add TestFile.txt
```

3. Ahora se verifica que el estado del fichero haya cambiado.

```
$ git status
```

Obtendrá una salida parecida a esta

```
[juand@juanpc structureExample]$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   TestFile.txt
```

4. Agregamos todos los cambios puestos en el *Work Tree* al head, por medio de un COMMIT, el argumento “-m” indica que ingresaremos un comentario<sup>9</sup>.

```
$ git commit -m "Ingresando mi primer cambio"
```

---

<sup>9</sup>En caso de no usar el argumento “-m” git abrirá por defecto un editor de texto (usualmente vim) donde se pueden especificar demás configuraciones (incluso el comentario). Por facilidad se recomienda siempre agregar el argumento “-m”

# Sección 5: Guía GitHub

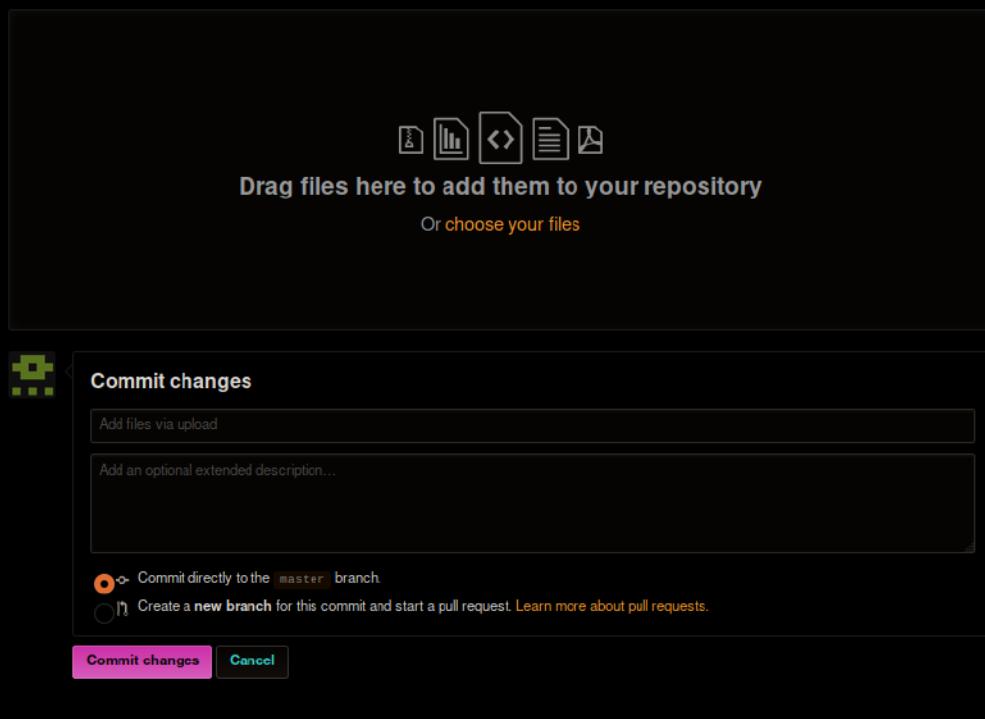
- Finalmente subimos los cambios al servidor de GitHub, este pedirá que ingrese su usuario y contraseña, asegúrese de que ingrese los datos correspondientes a la cuenta dueña del repositorio.

```
$ git push
```

Si no se muestra un mensaje de error, usted ha completado este paso satisfactoriamente

## Git con navegador web

- Ingrese a su repositorio, vaya al directorio en donde desea ingresar o añadir su fichero y/o directorio y de clic en el botón *Update*. Ingresará a esta interfaz



- Una vez allí, seleccione los ficheros y/o directorios, agregue un comentario y de en el botón “*Commit changes*”.

Es importante que se asegure que tenga seleccionado la opción “*Commit directly to the MASTER branch*”

# Sección 5: Guía GitHub

## 5. Haciendo cambios y moviendo ficheros

En este momento deberá tener su directorio del repositorio lo siguiente:

```
juan@juan-VirtualBox:~/GuiaGit$ ls  
laboratorios proyecto talleres TestFile.txt
```

Además, al ver el estado del directorio deberá aparecer lo siguiente:

```
[juand@juanpc structureExample]$ git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
nothing to commit, working tree clean
```

Si no es así, regrese al **Paso 4**

### Git con *Bash*

1. Abra con su editor de preferencia<sup>10</sup> el archivo *TestFile.txt* y realice cualquier cambio en el archivo.
2. Veamos como el estatus del repositorio ha cambiado

```
juan@juan-VirtualBox:~/GuiaGit$ git status  
On branch master  
Your branch is up-to-date with 'origin/master'.  
Changes not staged for commit:  
  (use "git add <file>..." to update what will be committed)  
  (use "git checkout -- <file>..." to discard changes in working directory)  
  
        modified:   TestFile.txt
```

**IMPORTANTE:** Este procedimiento es exactamente el mismo cuando desea subir el cambio de cualquier fichero o directorio de su repositorio a *GitHub*.

<sup>10</sup>Puede ser Emacs, Vim, Code, Notepad, etc.

# Sección 5: Guía GitHub

3. Agregue el cambio al *Head*

```
$ git add TestFile.txt
```

4. Realice el *commit* de los cambios

```
$ git commit -m "Modificando TestFile"
```

5. Luego, súbalos al servidor.

```
$ git push
```

6. Ahora movamos este archivo dentro de la estructura de directorios

```
$ git mv TestFile.txt ./talleres/
```

7. Veamos el status del repositorio cambio a

```
juan@juan-VirtualBox:~/GuiaGit$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    renamed:    TestFile.txt -> talleres/TestFile.txt
```

8. No se necesita agregar un movimiento, realice el *commit* de los cambios

```
$ git commit -m "Moviendo TestFile"
```

9. Finalmente, súbalos al servidor.

```
$ git push
```

# Sección 5: Guía GitHub

## Git con navegador web

1. GitHub cuenta con un editor simple de texto al que se accede dando clic en el archivo que se desea modificar. Los cambios son automáticamente actualizados en los repositorios.
2. Via browser no es posible mover ficheros o directorios.

## 6. Borrando ficheros

### Git con Bash

1. Borraremos *TestFile*. Para esto vaya al directorio donde se encuentra dicho fichero: <nombreDelRepositorio>/talleres
2. Ingrese los comandos:

```
$ git rm TestFile.txt
```

3. Veamos cómo se modificó el estatus de los directorios:

```
juan@juan-VirtualBox:~/GuiaGit/talleres$ git status
On branch master
Your branch is up-to-date with 'origin/master'.
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    deleted:    TestFile.txt
```

4. Realizamos el *commit*:

```
$ git commit -m "Eliminando archivo TestFile"
```

5. Finalmente, suba los cambios al servidor:

```
$ git push
```

# Sección 5: Guía GitHub

## Git con navegador web

- 1.** De clic sobre el archivo que desea borrar, y luego en el ícono de eliminar que está en la esquina superior derecha del editor.
- 2.** Vía Browser no es posible eliminar un directorio.

# ¿Alguna inquietud?

## CONTACTO

**Docente Mauricio Toro Bermúdez**  
**Teléfono:** (+57) (4) 261 95 00 **Ext.** 9473  
**Correo:** mtorobe@eafit.edu.co  
**Oficina:** 19- 627

Agende una cita con él a través de <http://bit.ly/2gzVg10>, en la pestaña **Semana**. Si no da clic en esta pestaña, parecerá que toda la agenda estará ocupada.



## Realizadores

**Este texto fue escrito y corregido  
por:**

**Mauricio Toro Bermúdez  
Luis A. Gallego  
Juan David Arcila Moreno  
Luisa Fernanda Alzate Sánchez**